On the Implementation of Kernel Numerical Algorithms for Computational Fluid Dynamics on Hypercubes

Tony F. Chan

November 1986

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS TR 86.23

RIACS
Research Institute for Advanced Computer Science

On the Implementation of Kernel Numerical Algorithms for Computational Fluid Dynamics on Hypercubes

Tony F. Chan

November 1986

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS TR 86.23

On the Implementation of Kernel Numerical Algorithms for Computational Fluid Dynamics on Hypercubes

Tony F. Chan[1]

RIACS Technical Report 86.20
November, 1986

[1]This work is supported by the Research Institute in Advanced Computer Science, NASA Ames Research Center and the Department of Energy under Contract DE AC02 81ER 10996 while the author is on sabbatical leave from Yale University. Current address: Dept. of Mathematics, UCLA, 405 Hilgard Av., Los Angeles, CA 90024. This paper is prepared for the proceedings of the Second Conference on Hypercube Multiprocessors, Knoxville, Tennessee, September 29 - October 1, 1986.

Abstract: We discuss some issues that arise in the implementation of numerical algorithms for computational fluid dynamics (CFD) on multiprocessor systems such as hypercubes. We identify several important kernel numerical algorithms from CFD that map well onto the hypercube architecture. We emphasize the importance of considering the optimal mapping for a collection of kernel algorithms used in an application program rather than just on individual optimal mappings. Several examples illustrating the tradeoffs between rearranging data to fit a particular kernel algorithm and using suboptimal mappings will be discussed.

1. Introduction

The field of computational fluid dynamics (CFD) has been a primary motivating force behind the recent interest in parallel computing. CFD demands lots of compute power, primarily due to the presence of wide scales of physical phenomenon that need to be modelled. Since CFD applications are already stretching the limits of sequential processing, many people believe that the computing power needed for major advances in CFD can only be delivered by parallel processing.

This trend has already been recognized for many years at CFD research centers such as the NASA Ames Research Center, which developed one of the earliest parallel computers, the ILLIAV IV. Recent developments there include the NAS project (Numerical Aerodynamics Simulation) one of whose goals is to bring the most powerful supercomputer at any given time to applications in CFD. Currently, it operates a 4 processor Cray 2 system. Looking into the future, NAS sees computers with massive parallelism. With this trend, the interplay between architecture and algorithms becomes more important. For this reason, the Research Institute for Advanced Computer Science (RIACS) has initiated a research program to bridge the gap. Efforts so far include studies of CFD on the MPP parallel computer and the Intel iPSC hypercube multiprocessor computer by John Bruno [2,3] and a workshop headed by Jack Dennis on the use of data flow architectures in CFD [1].

This article describes a preliminary study on the use of hypercubes to CFD. Our approach here is to identify and study kernel algorithms in CFD in detail in order to provide a scientific basis for implementation into future production codes. A main issue we study is the optimization of load balancing and communication cost by careful mapping of data (domain). We put a special emphasis on selecting optimal mappings for a collection of algorithms used in a typical application rather than on mappings for individual kernels. We also study the tradeoffs between the cost of using a suboptimal mapping and that of rearranging the data to obtain an optimal mapping. Several examples from CFD will be used to illustrate our points.

2. Kernel Algorithms in CFD

CFD can be classified by the various versions of the Navier Stokes equations that it solves: e.g. incompressible flows, compressible flows, transonic flows, turbulence modelling etc. Due to the different mathematical characteristics of the governing equations, these variants usually require different computational algorithms for their solutions. In this section, we identify some of the commonly used kernel algorithms in each case.

First of all, most methods require some sort of nearest neighbor mesh (NN-mesh) computations defined by the discretization stencil, such as for the computation of the residual and in relaxation methods.

In compressible flows, alternating direction implicit methods (ADI) are popular. These methods require the solution of block tridiagonal linear systems in each coordinate direction at each time step. A representative code is ARC3D developed at NASA Ames [19].

In incompressible flow, there is usually a Poisson equation to be solved at each time step (for the pressure in the primitive variable formulation and for the stream function in the stream function vorticity formulation), which often consumes most of the computation time. Thus a fast Poisson solver is an important kernel for these applications.

Spectral methods are becoming more widely used in CFD. In turbulence modelling [18], they are particularly predominant. Combined with their applications to other kernels such as fast Poisson solvers, this makes the fast Fourier transform (FFT) an extremely important kernel for CFD.

Transonic flows are usually modelled by equations which have different type (hyperbolic or elliptic) in different regions of the computational domain. Since the location of these type transitions depends on the solution itself, the discretization may change during the solution procedure. This feature leads to iterative methods (usually relaxation methods) which makes successive sweeps across the domain to update the solution. These iterative methods are often accelerated by techniques such as multigrid and preconditioned conjugate gradient methods, which are finding increasing application in other areas of CFD as well. The most well-known codes in this area are perhaps the series of codes starting from FLO52 by A. Jameson [13]. Recent versions of this code use the ADI method with multigrid acceleration.

3. Mapping of Algorithms to Hypercube Architecture

Part of the reason for the popularity of the hypercube architecture is the fact that many other topologies, such as meshes, trees, pyramids and butterflies, are embeddable in the hypercube topology [21]. By this we mean the graphs representing these other topologies can be mapped onto the binary n-cube graph with small dilation (the relative increase in the distance between vertices) and expansion (the relative increase in the total number of vertices) [20]. In practice, this implies that computations involving data flow graphs taking the form of the previously mentioned topologies can be carried out with minimum communication overheads on binary n-cube computers. In this section, we shall briefly discuss the problem of mapping some of the kernel algorithms in CFD identified in the last section onto the hypercube. These include FFT (butterfly), cyclic reduction (tree), multigrid (pyramid) and computations on NN-meshes.

3.1. FFT

We shall restrict our discussions to the radix-2 FFT algorithm. The data flow graph for the FFT is usually referred to as the "butterfly". Let the array elements be x_j, where the index j ranges from 0 to 2^d-1. These array elements are updated at each of d stages of the computation. At the i-th stage of the computation, the array element with index j must communicate with another element with index k, whose binary representation differs from that of j in the i-th most significant position. On a hypercube, the natural mapping is to map x_j to node number j of the cube [9]. To be mathematically precise, let $M:I \rightarrow N$ be the class of mapping functions, where I denotes the set of indices of the array elements and N the set of node numbers of the hypercube. Then the FFT mapping $f \varepsilon M$ is given by

$$f(j) = j. \tag{1}$$

With this mapping, it is easy to see that at every stage of the computation, the necessary communication will be between neighboring nodes. Moreover, all such communications at a given stage can be carried out in parallel. Each stage can be viewed as "collapsing" the hypercube in one of its coordinates. In fact, the hypercube is *isomorphic* to the butterfly of the same dimension.

Note that after the completion of the forward transform, the data elements are not arranged in the natural order. Rather, they are in what is known as bit-reversed order. This is unimportant if the inverse transform of the data array is to be computed next, perhaps after some computations on the transformed array itself, which is typical in many applications. If needed, the array elements can be permuted into natural order in d steps with only nearest neighbor communication [23].

3.2. NN Mesh

Another very common data flow graph is the nearest neighbor (NN) mesh, which occurs naturally in the solution of partial differential equations. We shall limit our discussion here to one dimensional meshes, since higher dimensional meshes can be easily built from tensor products of one dimensional ones[7,21]. If the array elements are denoted by x_j, then the NN-mesh graph with the x_j's as vertices contains all edges connecting a given vertex x_j with its nearest neighbors on the mesh. In one dimension, these are the two vertices x_{j-1} and x_{j+1}. (Throughout this paper, all indices are to be taken modulo 2^d, where d is the dimension of the hypercube.)

It is well-known that the NN-mesh graph with 2^d vertices can be mapped into a d-dimensional hypercube with no dilation or expansion via Gray codes. The natural mapping $m \varepsilon M$ for the NN-mesh to a d-dimensional hypercube is

$$m(j)=g_j, \tag{2}$$

where g_j is the j-th member of a d-dimensional Gray code. It is important to note that Gray codes are not unique but any d-dimensional Gray code will work here.

3.3. Multigrid

Multigrid algorithms can be viewed as methods for accelerating relaxation methods by performing extra iterations on a hierarchy of coarser grids in addition to the fine grid on which the solution is sought. On the finest grid, only NN-mesh computations are performed and therefore the tensor product Gray code mapping discussed in the last section can be used effectively. On the coarser grids, however, NN-mesh computations require communication between grid points that are increasingly becoming farther apart. Thus a little bit more care must be used to map the grid onto the hypercube so that coarse grid communications can be made efficient. In [6], it is shown that, if the grid coarsening factor is 2, then by using a special Gray code (namely the binary reflected Gray code (BRGC)) to map the finest grid to the hypercube, all coarse grid communications are exactly distance 2 apart (which is optimal), independent of the size of the grid or that of the hypercube. We emphasize that this is a special property of the BRGC and does not hold for many other Gray codes. Thus, the same mapping which is optimal for NN-meshes is also optimal for multigrid.

3.4. Cyclic Reduction

Odd Even Cyclic Reduction (CR) is one of the most efficient parallel algorithm for solving tridiagonal systems [12]. In the first stage of the algorithm, the odd numbered equations are used to eliminate the odd numbered unknowns in the neighboring even numbered equations, resulting in a new tridiagonal system of half the original size governing only the even numbered unknowns. Subsequent stages are recursive applications of the same principle. If the size of the system n is a power of 2, this leads to one equation in one unknown after $\log_2 n$ stages, which can be solved trivially. This is followed by a backsolving phase reversing the data flow of the forward elimination phase. The communication requirement is similar to the multigrid algorithm: starting from NN-mesh at the first stage, communications are between nodes that are at increasing powers of 2 apart. Therefore, it is easy to see that if the BRGC is used to map the equations to the hypercube, then all communications in subsequent stages are exactly at distance 2 apart [15], which is optimal.

4. Suboptimal Mappings Versus Data Rearrangement

In the last section, we saw that the optimal mapping for many kernel algorithms in CFD are easy to define and implement. However, what is often not appreciated is the fact that different algorithms require different mappings. Moreover, in applications where the same data set must be operated on by more than one algorithm, situations may arise where the optimal mappings for the individual algorithms are not compatible with one another. What is ideal for one could be disastrous for the others. In such situations, we have two alternatives: rearrange the data between algorithms or use only one mapping which may be suboptimal for some of the algorithms. The suboptimality could be either due to communication delays or to imperfect load balancing. The tradeoff between these two approaches depends on the relative frequency each algorithm in the collection is executed. In this section, we shall illustrate these issues by several examples in CFD.

4.1. The Incompatibility Of the FFT and the NN-Mesh Mappings

The most natural mapping for the FFT algorithm, namely f in equation (1), turns out to be inefficient for NN-mesh computations. If the data array x_j is mapped onto the hypercube using the FFT mapping f, then a nearest neighbor computation on the same data array requires communication over a distance d (equal to the diameter of the hypercube) for some nodes, e.g. between node $2^{d-1}-1$ and node 2^{d-1}. Thus, as far as the NN-mesh computation is concerned, the FFT mapping f is the *worst* possible mapping. For higher dimensional hypercubes, this may cause a significant reduction in the efficiency of algorithms using the nearest neighbor data flow graph.

On the other hand, the NN-mesh mapping m in equation (2) can be made suitable for FFT computations. To see this, note that the FFT butterfly involves pairs of array elements separated by strides of diminishing powers of 2 and thus if the array elements are mapped to the hypercube using a BRGC, then the communications occur between nodes at most a distance two apart. This is optimal because a distance of one would create a cycle of odd length in the hypercube (the power of 2 stride plus the nearest neighbor connection), which is impossible [21].

Alternatively, one can consider rearranging the data before the execution of each algorithm according to the most natural mapping for that algorithm, especially if the algorithm is to be executed many times. It turns out that one can convert from a BRGC NN-mesh mapping to a FFT mapping (and vice versa) with $d-1$ nearest neighbor exchanges [5,15], which is comparable with the communication cost of one FFT computation. Thus if many FFTs are to be performed, then it may pay to carry out the conversion to the optimal FFT mapping. For a detailed analysis of which mapping is best for a particular application and the tradeoffs between rearranging data and using a suboptimal mapping, the reader is referred to [5].

Applications employing both the FFT and the NN-mesh data flow graphs are very common in CFD. A finite difference discretization of the Navier Stokes equation often produces a nearest neighbor stencil. Thus the NN-mesh mapping is natural for computation of residuals. On the other hand, if a spectral method or a fast Poisson solver is used to solve the difference equations, then the FFT mapping is called for.

4.2. The Parallel Exchange Multigrid Algorithm

Sometimes it may be beneficial to rearrange data even between different parts of the *same* algorithm. In the parallel implementation of multigrid methods on hypercubes as described in Section 3.3, by using a BRGC NN-mesh mapping of the finest grid, communication on coarser grids are at distance exactly 2 apart. If many NN-mesh operations are to be performed on a coarse grid, then one can consider rearranging the data on it to reduce this distance to 1. In [6], it is shown how this can be achieved by a simple exchange communication between certain neighboring nodes on the fine grid before starting computations on the coarse grid. A similar exchange must be used when coming back from a coarse grid to a fine grid. This remapping thus pays if more than 2 smoothing sweeps are performed on the coarse grid. See [8] for more details on the implementation of this algorithm.

4.3. Transpose in ADI Methods

Another example of the tradeoff between using a suboptimal mapping and rearranging data to an optimal mapping is in the parallel implementation of Alternating Direction Implicit Methods (ADI) on hypercubes [16]. We shall restrict our discussion to two dimension regions. Each iteration of the ADI method consists of solving independent tridiagonal systems on grid lines in each of the two coordinate directions. If the grid is partitioned into groups of grid lines along one coordinate direction and each group mapped onto the hypercube according to the BRGC, then the solution of the tridiagonal systems in this chosen direction are all within the same processor and requires no communication. The solves in the other direction can be solved by the cyclic reduction algorithm described in Section 3.4, with communication between processors at distance 2 apart after the first reduction stage. By a similar exchange operation as in the multigrid algorithm described earlier in Section 4.2, this distance can be reduced to one [15]. By a more costly transpose operation of the grid, this distance can be further reduced to zero (solves within the same processor). Which alternative is best is still an open research problem and probably depends on the sizes of the grid and the hypercube and the relative cost of computation and communication [11,17,22].

4.4. Load Balancing in Parallel FFT

The previous examples show situations where remapping can reduce the communication cost of an algorithm. Sometimes, remapping can improve load balancing. Consider the parallel implementation of the Gentleman-Sande version of the FFT [24]. The original sequence is divided into contiguous blocks and each block mapped onto the processors of the hypercube using the FFT mapping f discussed earlier. At each stage, two kinds of computations must be performed: (1) the sum of two intermediate transformed values and (2) their difference, multiplied by appropriate powers of certain roots of unity. In the early stages, these two computations are performed on neighboring processors requiring communication whereas in the latter stages, they are performed in the same processor requiring no communication. In [10], it is pointed out that in the early stages the computations are not perfectly balanced: some processors must compute (or look up) the roots of unity and perform a complex multiplication whereas others do not. This imbalance can be eliminated by permuting the data at every stage so that only local FFTs of the same size are performed in each processor. For more details, the reader is referred to [10].

4.5. Particle in Cell Methods

Another example of remapping to improve load balancing is in certain parallel implementation of particle in cell methods [4]. These methods typically consists of two stages: the solution of a Poisson equation for the field on a regular grid and then "particle pushing" of the individual particles. The Poisson equation is usually handled by a "fast solver" which prefers a regular mapping, e.g. the tensor product BRGC NN-mesh mapping. However, if the particles are distributed nonuniformly in the computational domain, then the particle pushing phase prefers a nonregular mapping which would produce better load balancing in the sense of having roughly the same number of particles in each processor. Again the tradeoff is between the cost of remapping and that of using a suboptimal mapping.

5. Concluding Remarks

A typical CFD program contains several kernel algorithms working on the same data. One of the main purposes of this article is to emphasize the need to go beyond studying the optimal implementation of the individual algorithms and look at the whole collection as a whole. The reason is simply that what is best for one algorithm may not be optimal for the others. Very often, a remapping of the data to perfectly suit an algorithm should be considered. The tradeoff is between the cost of the remapping and the benefits of reduced communication and better load balancing. We have shown in this article several such examples arising naturally in CFD applications. The optimal choice of mapping depends on many factors, such as the frequency the algorithms are to be executed and the ratio of the arithmetic speed to the communication speed.

Reference

[1] G.B. Adams, R.L. Brown, P.J. Denning, *Report on an Evaluation Study of Data Flow Computation*, RIACS Technical Report 85.2, Research Institute in Advanced Computer Science, 1985.

[2] J. Bruno, *Final Report on the Feasibility of Using the Massively Parallel Processor for Large Eddy Simulation and other Computational Fluid Dynamics Applications*, RIACS Technical Report 84.2, Research Institute in Advanced Computer Science, 1984.

[3] J. Bruno, *Report on the Feasibility of Hypercube Concurrent Processing Systems in Computational Fluid Dynamics*, RIACS Technical Report 86.7, Research Institute in Advanced Computer Science, March 1986.

[4] C. Catherasoo, *The Vortex Method on a Hypercube*, paper presented at the 2nd Conference on Hypercube Multiprocessors, Knoxville, Tennessee, September 29 - October 1, 1986.

[5] T.F. Chan, *On Gray Code Mappings for Mesh-FFTs on Binary N-Cubes*, RIACS Technical Report 86.17, Research Institute in Advanced Computer Science, September, 1986.

[6] T.F. Chan, Y. Saad, *Multigrid Algorithms on the Hypercube Multiprocessor*, IEEE Trans. on Comp., Nov., 1986, pp. 969-977.

[7] T.F. Chan, Y. Saad, M.H. Schultz, *Solving Elliptic Partial Differential Equations on Hypercube Multiprocessors*, Proceedings of 1st Conference on Hypercube Multiprocessors, ed. M. Heath, pp. 196-210, SIAM, Philadelphia, 1986.

[8] T.F. Chan, R. Tuminaro, *Implementation of Multigrid Algorithms on Hypercubes*, RIACS Technical Report 86.30, Research Institute in Advanced Computer Science, November, 1986. To appear in the proceedings of the 2nd Conference on Hypercube Multiprocessors, Knoxville, Tennessee, September 29 - October 1, 1986.

[9] G. Fox, S. Otto, *Decomposition of Scientific Problems for Concurrent Processors*, Physics Today, May, 1984.

[10] G. Fox, W. Furmanski, *Some Highlights of Hypercube Research at Caltech*, paper presented at the 2nd Conference on Hypercube Multiprocessors, Knoxville, Tennessee, September 29 - October 1, 1986.

[11] C.T. Ho, S.L. Johnsson, F. Saied, M.H. Schultz, *The Three Dimensional Wide Angle Wave Equation, Tridiagonal Systems and the Intel iPSC*, paper presented at the 2nd Conference on Hypercube Multiprocessors, Knoxville, Tennessee, September 29 - October 1, 1986.

[12] R.W. Hockney, C.R. Jesshope, *Parallel Computers*, Adam Hilger Ltd., Bristol, 1981.

[13] A. Jameson, *Solution of the Euler Equations for Two Dimensional Transonic Flow by a Multigrid Method*, Appl. Math. and Comp., vol.13, pp.327-355, 1983.

[14] L.S. Johnsson, *Odd-Even Cyclic Reduction on Ensemble Architectures*, Research Report YALEU/DCS/RR-339, Yale Univ., Dept. of Comp. Sci., Oct. 1984.

[15] L.S. Johnsson, *Communication Efficient Basic Linear Algebra Computations on Hypercube Architecture*, Research Report YALEU/DCS/RR-361, Yale Univ., Dept. of Comp. Sci., Sept. 1985.

[16] L.S. Johnsson, Y. Saad, M.H. Schultz, *Alternating Direction Implicit Methods on Multiprocessors*, Research Report YALEU/DCS/RR-382, Yale Univ., Dept. of Comp. Sci., Oct., 1985.

[17] D. Lim, R.V. Thanakij, *Alternating Direction Implicit Methods on a Hypercube*, paper presented at the 2nd Conference on Hypercube Multiprocessors, Knoxville, Tennessee, September 29 - October 1, 1986.

[18] R.S. Rogallo, *An ILLIAC Program for the Numerical Simulation of Homogeneous Incompressible Turbulence*, NASA Document No. NASA TM-73, 203, 1977.

[19] T.H. Pulliam, J.L. Steger, *Implicit Finite Difference Simulations of Three Dimensional Compressible Flow*, AIAA J., vol. 18, p.159, 1980.

[20] A. Rosenberg, *Data Encoding and Their Costs*, Acta Inform. 9 (1978), pp.273-292.

[21] Y. Saad, M.H. Schultz, *Some Topological Properties of the Hypercube Multiprocessor*, Research Report YALEU/DCS/RR-428, Yale Univ., Dept. of Comp. Sci., October, 1985.

[22] F. Saied, *ADI Methods for Schrodinger's Equations on Hypercubes*, paper presented at the 2nd Conference on Hypercube Multiprocessors, Knoxville, Tennessee, September 29 - October 1, 1986.

[23] J. Salmon, *private communication*. See also Chapter 8 of the book "*Solving Problems on Concurrent Processors*" by G. Fox et al, to be published.

[24] P. N. Swarztrauber, *Multiprocessor FFTs,* paper presented at the International Conference on Vector and Parallel Computing, Loen, Norway, June 2-6, 1986.